ATCS

# HLM20 高级语言模块
# 使用手册

Techwayson 德 维 森

# 目　录

# 硬

# 件

# 篇

# 第一章 高级语言模块介绍

## 1-1 产品说明

传统控制器所连接的输入\输出装置不外乎是属开关式(ON/OFF)或模拟式(Analogue) 因此所提供的输入\输出界面大都是以能连接此二种类型的信号为主。时至今日，由于工厂自动化程度的需求日高控制器担负的工作角色亦日趋复杂，其所须连接的装置不再仅限于上述的两种类型。在增加的信号类型中有一种是属串行(Serial)的传输信号。由于传递的信息皆利用ASCII码来传送;因此一般通称这类的装置为ASCII装置。例如条形码输入装置(Barcode reader)，计算机字幕机(Moving sign)等。

这种装置具有两项特色:

(1)以串行的方式传递信号

(2)不同装置各有不同的信息格式(Format)

为了能与不同的ASCII装置沟通，德维森控制器提供了高级语言模块(High Level Language Module，以下简称为HLM20模块）。在模块上提供了连接ASCII装置所需的硬件界面，以及软件操作环境。使用者可依据实际使用的ASCII装置信息格式来编写应用程序,以达到与ASCII装置沟通的目的。

## 1-2 特色

**HLM20模块具以下的特色**

--处理能力

- 采用高性能的16位CPU，执行速度快。
- 应用内存容量120K Bytes，断电后资料可保留一年。
- 测试应用程序时，可在RAM上执行实际运转时才采用EEPROM。

--提供与ASCII装置连接所需的硬件

- 两个串行通讯端口。
- 各通讯端口可独立设定为RS232或RS422。
- 模块面板上提供传送／接收状态指示灯，便于掌握传输状况。

--便利的软件开发环境

- 利用个人计算机发展应用软件。
- 采高级的C语言编写应用软件，程序易于维护。
- 提供易于使用的read/write 控制器的状态函数，以及串行通讯驱动函数。
- 提供多任务作业的函数供使用者调用,使得程序编写更加容易。

## 1-3 规格

<table>
<tr><td colspan="2">接口</td><td>RS-232C/422</td></tr>
<tr><td colspan="2">协议</td><td>异步通讯，全/半 双工</td></tr>
<tr><td colspan="2">通道</td><td>2ch</td></tr>
<tr><td colspan="2">应用程序容量</td><td>120 K bytes</td></tr>
<tr><td colspan="2">数据缓冲区</td><td>512 bytes</td></tr>
<tr><td colspan="2">传输数率</td><td>1200,2400,4800,9600,19200,38400 bps</td></tr>
<tr><td rowspan="3">数据格式</td><td>数据位</td><td>7,8</td></tr>
<tr><td>奇、偶校验位</td><td>无、奇、偶</td></tr>
<tr><td>停止位</td><td>1,2</td></tr>
<tr><td colspan="2">隔离方式</td><td>光电隔离</td></tr>
<tr><td colspan="2">参数设定</td><td>软件设置</td></tr>
<tr><td colspan="2">最大电流消耗</td><td>Max.450mA</td></tr>
<tr><td colspan="2">附件</td><td>DB9 （针）连接器 x 2</td></tr>
<tr><td colspan="2">操作指示</td><td>电源指示灯（LED），TX/RX 指示灯（LED）</td></tr>
<tr><td colspan="2">外部连接</td><td>DB9 （孔）连接器</td></tr>
<tr><td colspan="2">重量</td><td>300 克</td></tr>
</table>

# 第二章 安　　装

## 2-1　机架

HLM20 模块适用于 ATCS PPC11 系列 PPC 及 ATCS PPC22 系列软逻辑控制器。应用时可插于任何机架的 I/O 插槽上.

## 2-2　模块面板说明(参考图一)

1. ACT(动作指示灯)
   当此模块在执行监督程序时闪烁较慢(约 0.2HZ),在执行应用程序时闪烁较快(约 5HZ)。
2. SEL(与控制器沟通指示灯)
   当此灯亮时，表示 HLM20 模块与控制器正进行信息沟通，当控制器与 HLM20 模块沟通时此灯应会闪动。
3. TX1(COM1 资料传送指示灯)
   当此灯亮起时，表示 1 号通讯端口正在传送数据。
4. RX1(COM1 资料接收指示灯)
   当此灯亮起时，表示 1 号通讯端口有数据传入。
5. TX2(COM2 资料传送指示灯)
   当此灯亮时，表示 2 号通讯端口正在传送数据。
6. RX2(COM2 资料接收指示灯)
   当此灯亮起时，表示 2 号通讯端口有数据传入。
7. Toggle SW
   **具有以下两种功能:**
   (1)用于应用程序侦错.当应用程序利用监督程序的”G”指令执行,若改变此开关的位置,会强迫跳离应用程序执行监督程序。
   (2)防止 EEPROM 误写。此开关位置于下方时,EEPROM(可用来存放应用程序)无法烧录.当利用监督程序的”P”指令,将应用程序存于 EEPROM 时,须将此开关置于上方位置才有作用。
8. COM1 串行通讯端口
   此端口有两种用途在 PROG MODE 时利用此端口将应用程序加载,在 RUN MODE 时可拿来做应用连接 ASCII 装置。可用 DIP SW 或软件设定成 RS232 或 RS422。
9. COM2 串行通讯端口
   此端口为标准的应用端口，可连接 ASCII 装置。可用 DIP SW 或软件设定成 RS232 或 RS422。

## 2-3 模块背面开关说明(参考图二)

1. DIP SW1

   ON:COM1 使用 RS422 传输。

   OFF:COM1 使用 RS232 传输。

2. DIP SW2

   ON:COM2 使用 RS422 传输。

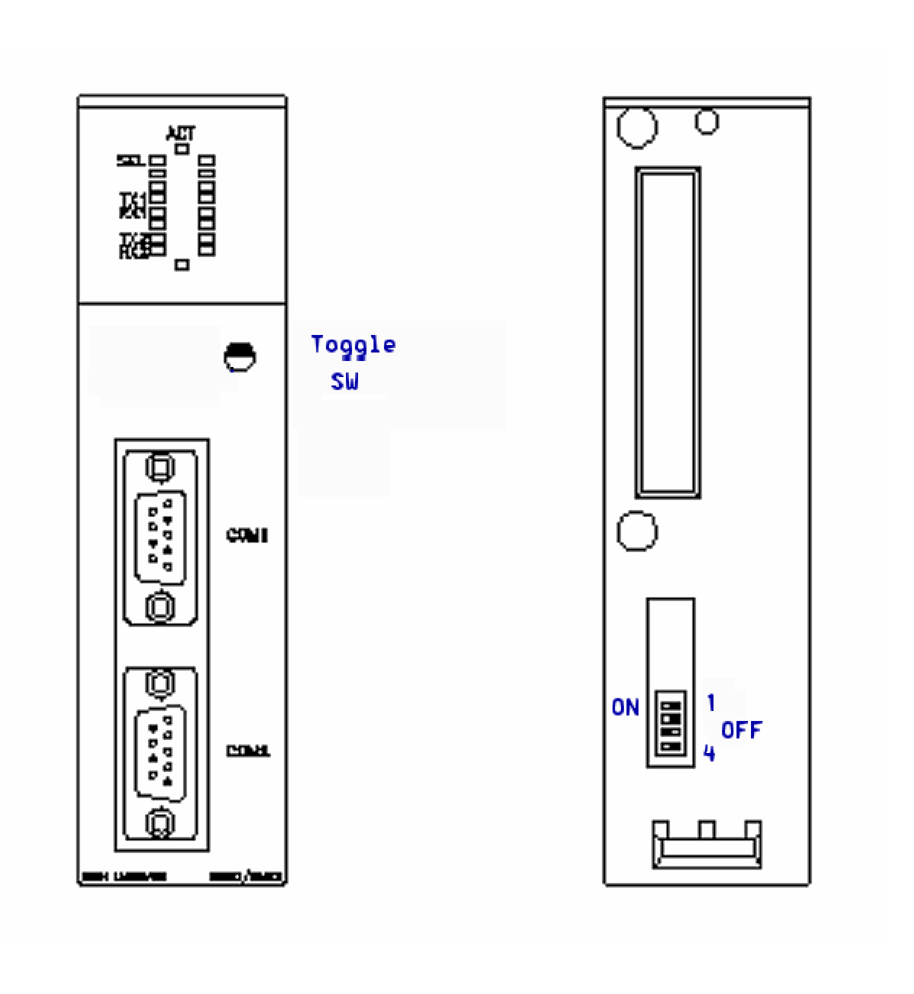
   OFF:COM2 使用 RS232 传输。

3. DIP SW3

   HLM20 模块有两种工作状态,一种为执行监督程序(PROG MODE)状态,在此状态下时,HLM20 模块等待由 COM1 与 PC 联机,另一种状态是执行使用者程序(RUN MODE) 。当开机时,如果此开关置于 OFF 位置则自动执行监督程序(PROG MODE),等待联机,反之如置于 ON 则自动执行使用者应用程序。

4. DIP SW4

   系统保留未使用。




图一 HLM20 面板图　　　　图二 HLM20 背面

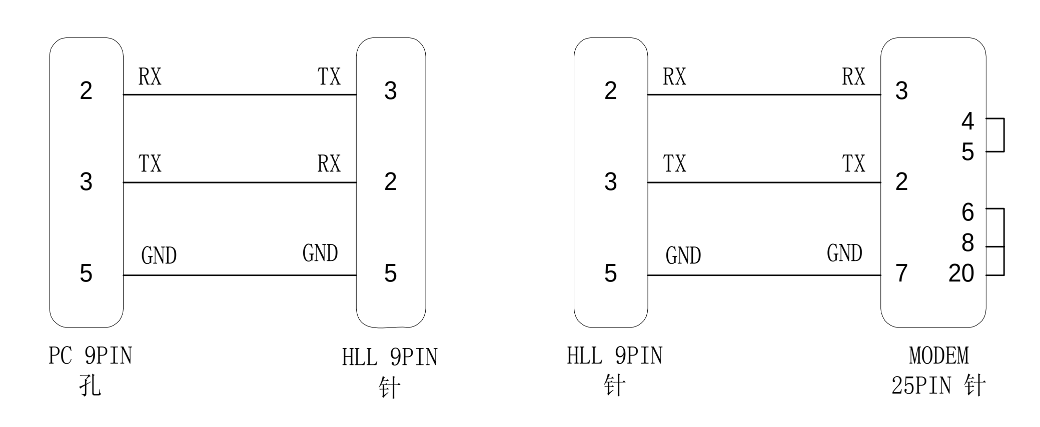
## 2-4　连接器脚位定义

1.连接器

HLM20 模块上有两个应用连接器，即串行通讯端口 1 与 2，此二端口均由 9 PIN 的 D 型母接头所构成，各 PIN 的接脚定义如下表所示：

| 1 | RX-(422) | 4 | TX-(422) | 7 | NC |
|---|---|---|---|---|---|
| 2 | RX(232) | 5 | GND(232) | 8 | TX+(422) |
| 3 | TX(232) | 6 | RX+(422) | 9 | RTS(232) |

2.应用接线图




图三　HLM20 应用接线图例

## 2-5　安装程序

1.检查模块背面的 DIP SW 位置是否正确，如果希望一开机即立刻执行应用程序则须将 SW3 置于 ON 位置；反之则置于 OFF 位置。

2.将 HLM20 模块插入机架，执行此步骤时须注意以下几点：

a.机架上的电源是否已关掉，如未关掉则须先关掉后才能插入。

b.插入后利用十字起子将模块上的螺丝旋紧，确保模块与机架间的良好接触。

c.将焊好的通讯线装好并锁上。

# 软

# 件

# 篇

# 第一章 高级语言模块软件开发环境

## 1-1 软件开发工具

1. 硬件

   IBM 兼容个人计算机 PC AT 以上，内存容量最好大于 256K；并且须包含至少一个的 RS232 通讯端口。

2. 软件

   a. DOS 3.0 以上操作系统。

   b. 德维森提供的 HLM20 模块发展装置软件内含：

      Hpcmoni.exe　　侦错工具程序
      mkapp.bat　　制作应用程序(置于 RAM)的批处理文件
      mkrom.bat　　制作应用程序(置于 RAM)的批处理文件
      start1.obj
      startup.obj　　目的文件用来连结应用程序与 HLM20 内建的函数
      stdio.h　　header 文件用于 I/O 函数的定义
      api.h　　header 文件用于 API 函数的定义
      ..
      ..
      ..

   c. 使用者须自备的软件- 68K C cross Compiler(美国 Microtec 公司的产品)内含

      mcc68k.exe　　编译程序
      asm68k.exe　　组译程序
      lod68k.exe　　连结程序
      math.h
      string.h　　引入挡
      ctype.h
      mcc68kab.lib

## 1-2 软件开发架构

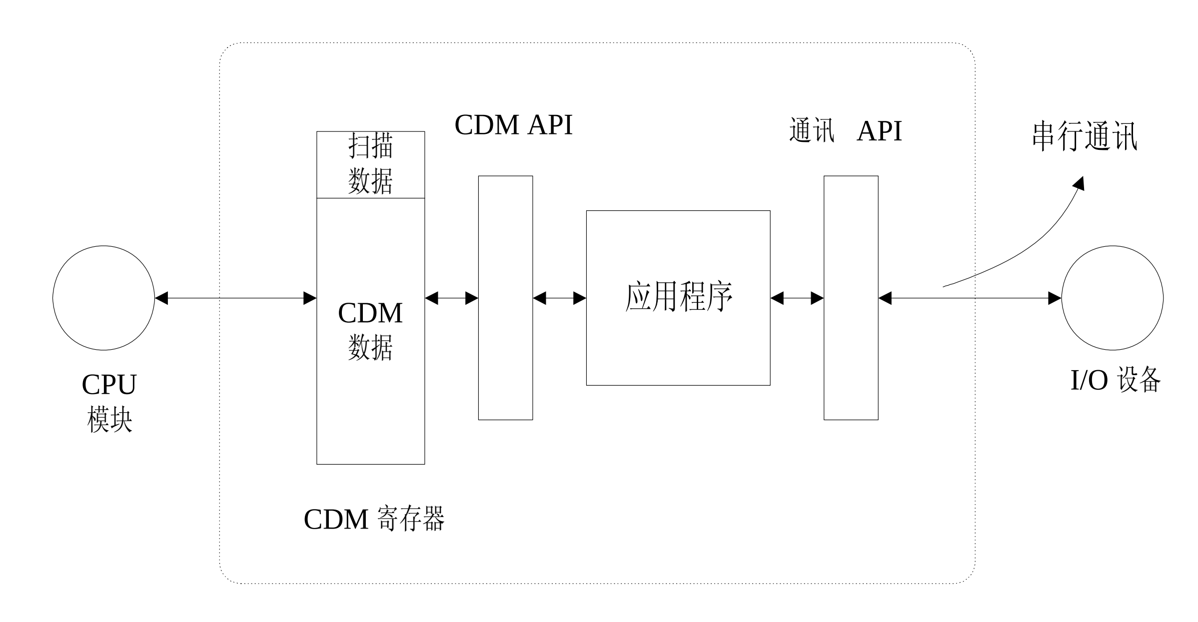



图 1 HLM20 模块架构图

HLM20 模块提供 CDM(Common Data Memory)与控制器沟通,软件部份提供 CDM API 及 Communication API 供使用者编写应用程序。

# 1-3　软件开发流程

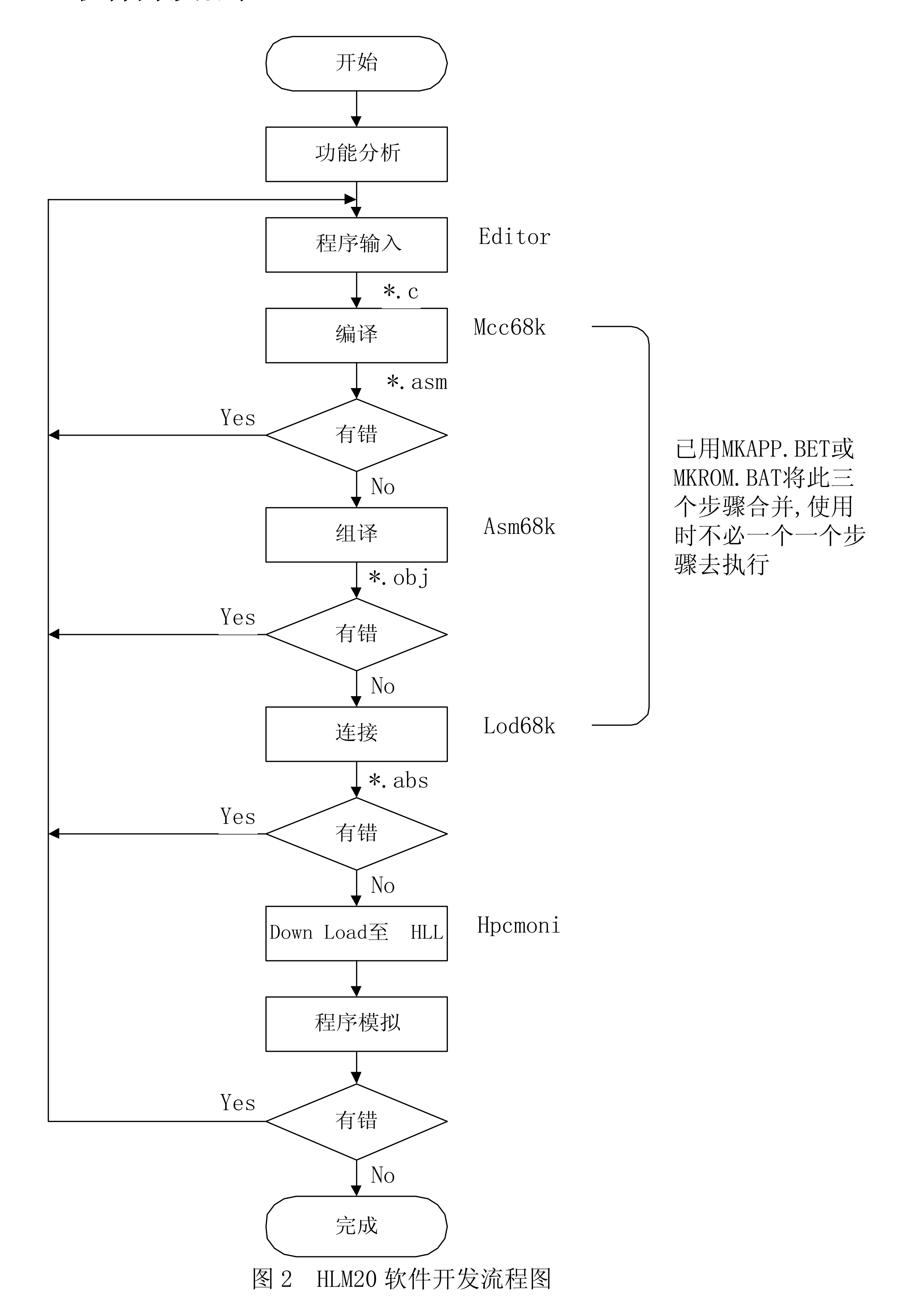



图 2　HLM20 软件开发流程图

程序开发步骤如下：

1. 利用编辑程序(例如 PE2)将用 C 语言写好的应用程序输至计算机。在此步骤产生的结果是一个文件名为*.C 的文字文件。
2. 建立一个 project file 内含 1 的主文件名，在此步骤产生的结果是一个文件名为*.prj 的文字文件。
3. 透过mkapp.bat的批处理文件,它会依照*.prj的内容一一去调用mcc68k编译程序(Compiler)将步骤 1 的文字文件转换为文件名为*.src 的文字文件。在调用 asm68K 组译程序(Assembler)将前面产出的汇编程序，转成文件名为*.obj 的目的文件。最后调用 lod68K 连结程序(LInker)将前面产出的目的文件与 HLM20 模块提供目的文件连结起来，最后产生一个文件名为*.ABS 的绝对文件。

   这一步骤是执行应用程序，以测试程序是否正确，利用 Hpcmoni.exe 直接将*.ABS 绝对文件传至 HLM20 模块上测试。

以下举一实例来说明：

假设使用者写了一应用程序，此一应用程序存放在两个文字文件内，即 TEST1.C 与 TEST2.C 其内容分别为下：

TEST1.C

```
#include <stdio.h>
#include <api.h>

struct BD_DATA {
UB      sub_id ;           /* Application board ID */
UB      di_gp_cnt ;        /* DI 16 point word group cnt,0~8 */
UB      ri_gp_cnt ;        /* RI 16 point word group cnt,0~8 */
UB      do_gp_cnt ;        /* DO 16 point word group cnt,0~8 */
UB      ro_gp_cnt ;        /* RO 16 point word group cnt,0~8 */
                           /* DI+RI+DO+RO <= 16 */
void    *scan_data_ptr;    /* pointer to the first variable of
                              scan data */
} bd_dptb = { 1,0,0,0,0,NULL};
```

至少有一文件内需有一个 bd_dptb 的宣告,

```
main          ←    至少有一文件内需有一个名为 main 的函数，HLM20 模
{                  组会从 main 函数开始执行
  unsigned short i,j;
      •
      •
```

```
    •
  XYZ();
    •
    •
  }
test2.c
XYZ()
{
    •
    •
    •
  return;
}
```

建立一个 project file 文件名为 test.prj ， 内含要连结在一起的 C CODE 文件名,其内容如下:

```
test.prj
test1
test2
```

在建好以上文件后,在使用 MKAPP.BAT 或 MKROM.BAT 来进行编译组译连结的动作。

执行方式如下:

**方式 1：**MKAPP TEST，执行时未变动的 C 文件不进行 COMPILER，执行以上指令，如果程序没有错误应该会产生 TEST.ABS 的文件。再利用 Hpcmoni.exe 执行 Down Load 侦错的动作。如果动作有错则修改程序再用 MKAPP 或 MKROM 来产生 ABS 文件。

**方式 2：**MKAPP TEST 0， 同方式 1，不同的是增加一个‘0’的参数，此方式会将 prj 内所有的 C 文件重新 Compile 一次，不论其是否有变动。

## 1-4　高级语言模块提供的软件功能

HLM20 模块内提供的软件功能主要分两类，一种是提供应用程序与控制器沟通的能力，一种是提供应用程序与外界沟通的(串行通讯端口)，这两种服务都是透过函数的调用来进行,另外也提供辅助函数库及多任务作业函数库供使用者调用。

### 1-4-1　与控制器沟通

所谓与控制器沟通是指应用程序可以读取控制器内的状态，或者是更改控制器的状态。

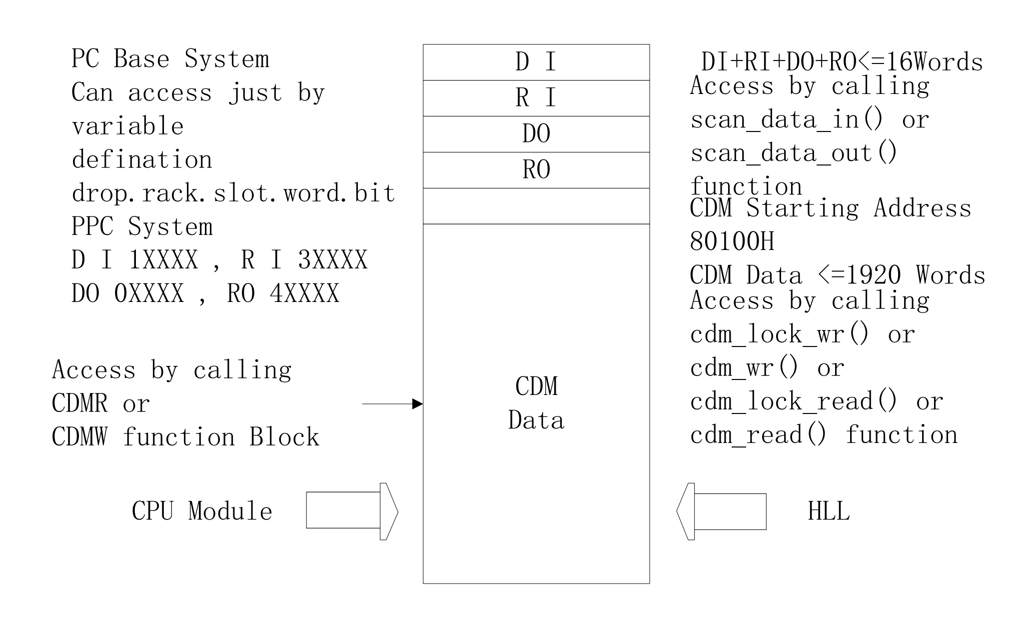



图 3　控制器与 HLM20 模块数据交换示意图

在此提供二种方式与控制器沟通,CDM Data 与 SCAN Data。

1. SCAN Data.
   a. 定义变量与控制器的状态关系。
      (参考 scan_data_in(),scan_data_out() function).
   b. 利用 scan_data_in()或 scan_data_out()函数来取得或设定控制器的资料。
2. CDM Data

   HLM20 模块提供 1920 个 WORD 的 CDM 数据区,供 HLM20 模块与控制器沟通用.可以用 cdm_lock_read(),cdm_read(),cdm_lock_wr(),cdm_wr()等函数来存取 CDM 的资料。

## 1-4-2　驱动串行通讯端口

HLM20 模块提供的串行通讯功能主要分三类分述于下：

1. 通讯参数的设定：
   利用 com_init 函数可设定传输速率，同位检出方式，字符长度　停止位长度等。
2. 信息的接收
   HLM20 模块提供 255 个字符的接收缓冲区，使用者透过 com_getc 来读取接收的资料，由于接收的信息皆是以中断的方式存于缓冲区，编写应用程序时比较不必担心会有资料丢掉(over run)的现象。
3. 信息的传送
   HLM20 模块提供了 255 个字符的传送缓冲区，使用者透过 com_puts 或 com_putb 等函数来传送资料，由于缓冲区的设置，应用程序不必浪费时间在等待信息传送上，可加速程序的执行。

## 1-4-3　辅助函数库

HLM20 提供软件中断，设定定时器及检查定时器的函数，供使用者除错用。

## 1-4-4　多任务作业函数库

HLM20 提供有关多任务作业的函数，供使用者更容易发展应用程序。

# 第二章
# 程序库内各函数的功能说明

HLM20 模块上提供的函数有两类，一种是标准的 C 语言函数，另一种则是与 I/O 有关的特殊函数

## 2-1　标准函数

HLM20 模块上提供的标准函数是依 Micortec C cross compiler 所定义的功能来加以编写，详细的函数说明请参阅 Microtec 的手册，以下仅列出 HLM20 模块提供的函数：

1. 字符处理函数

| Function | Difinition |
|---|---|
| atof | Convert ASCII to floating point |
| atoi | Convert ASCII to integer |
| atol | Convert ASCII to long integer |
| atoa | Convert integer to ASCII |
| itostr | Convert integer to string |
| itoa | Convert long integer to ASCII |
| itostr | Convert long integer to string |

2. 数学函数(Arithmetic function)

| Arithmetic | Definition |
|---|---|
| abs | Compute absolute value of an integer |
| acos | Compute arc cosine of an number |
| asin | Compute arc sine of a number |
| atan | Compute arc tanget of a number |
| atan2 | Compute arc tangent (overflow protection) |
| cos | Compute cosine of an number |
| cosh | Compute hyperbolic cosine of a number |
| erf | Compute errpr function |
| erfc | Complimentary error function |
| exp | Compute number to power of E |
| fabs | Compute absoluta value of a number |
| jn | Compute bessel function ‘n’ |

| | |
|---|---|
| j0 | Compute bessel function 0 |
| j1 | Compute bessel function 1 |
| log | Compute natural logarithm of a number |
| log10 | Compute logarithm of a number at base 10 |
| pow | Compute x raised to the power y |
| qsort | Quick sort algorithm |
| rand | Raturn a positive pseudo-random number |
| sin | Compute sine of a number |
| sinh | Compute hyperbolic sine of a number |
| sqrt | Compute square root of a number |
| srand | Set the seed of the random number generator |
| tan | Compute tangent of a number |
| tanh | Compute hyperbolic tangent of a number |

## 2-2 特殊函数

HLM20 模块提供的特殊函数计有以下的四类：

(1)串行通讯函数
(2)与控制器沟通函数
(3)辅助函数
(4)多任务作业函数

第一类函数有：

com_init(),com_getc(),com_putb(),com_puts(),com_status(),
go_rcv(),stop_rcv(),cnfg_232_422(),cnfg_232(),cnfg_422(),

第二类函数有：

cdm_lock_read(),cdm_read(),cdm_lock_wr(),cdm_wr(),
scan_data_in(),scan_data_out()

第三类函数有：

timer(),timer_status(),set_timer(),chk_timer(),led_service()
,
brk(),system(),crc16()

第四类函数有：

Create_task(),yield_task(),Task_get_pdata(),
Task_set_priority(),Suspend_task(),Resume_task(),preemtp_on(
),preemtp_off(),Delay(),Delay_until(),Get_timer()

(1)串行通讯函数

| | |
|---|---|
| com_init() | 设定通讯 port 的通讯参数 |
| com_getc() | 从 Com port 取得一个字符 |
| com_putb() | 将一个 BLOCK 资料从 Com port 送出 |
| com_puts() | 将一个字符串从 Com port 送出 |
| com_status() | 取得 Com port 的状态 |
| go_rcv() | Com port 开始从线上接收资料 |
| stop_rcv() | Com port 停止从线上接收资料 |
| cnfg_232_422() | 依 DIP SW 设定决定 Com port 使用 RS232 或 RS422 传输资料 |
| cnfg_232() | 强制 Com port 使用 RS232 传输资料 |
| cnfg_422() | 强制 Com port 使用 RS422 传输资料 |

■摘要

```
#include <api.h>
short com_init(port_no,compbk)
short port_no;   1->COM1 , 2->COM2
```

■说明

设定通讯 port 的通讯参数,包括 BAUD RATE,PARITY,DATA BITS,STOP BITS 等等.

■传回值

0 : 表示参数设定成功.
-1 : 表示参数设定有错.

■批注

```
struct COM_PMBK  {
    char        baud;
    /* 1:150,2:300,3:600,4:1200,5:2400,6:4800,7:9600,8:19200
     ,9:38400 ,COM1 COM2 不能同时设定 19200 与 38400*/
    char        parity;   /* 0:NONE , 1:ODD , 2:EVEN */
    char        data_bit; /* 6 or 7 or 8 */
    char        stop_bit; /* 1 or 2 */
    char        tx_ctrl;  /* 系统保留一律为 0 */
    char        rx_ctrl;  /* 系统保留一律为 0 */
    char        com_mode; /* 系统保留一律为 0 */
    char        tmo_mode; /* 系统保留一律为 0 */
    char        tmo_cnt;  /* 系统保留请勿设定 */
};
```

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

struct COM_PMBK  {
    char   baud;
    char   parity;
    char   data_bit;
    char   stop_bit;
    char   tx_ctrl;
    char   rx_ctrl;
```

```
    char   com_mode;
    char   tmo_mode;
    char   tmo_cnt;
};
struct COM_PMBK compbk ;

main()
{
        short i;
        char msg[100];
        compbk.baud = 8;     /* 19200bps */
        compbk.parity = 2;   /* EVEN */
        compbk.data_bit = 8; /* 8 BIT */
        compbk.stop_bit = 1; /* 1 BIT */
        compbk.tx_ctrl = 0;
        compbk.rx_ctrl = 0;
        compbk.com_mode= 0;
        compbk.tmo_mode= 0;
        i=com_init(2,&compbk);
        if(i==0) sprintf(msg,"com2 initial ok.");
        else sprintf(msg,"com2 initial not ok.");
        com_puts(1,msg);
}
```

■摘要

```
#include <api.h>
short com_getc(port_no)
short port_no;   1->COM1 , 2->COM2
```

■说明

取出由传输 port 接收外界所传来置于 RX_BUFFER 内一个字节的数据.

■传回值

-1:表示 RX_BUFFER 内无任何资料.

其它值:高字节(HIGH BYTE)表示接收数据的状态.

低字节(LOW BYTE)表示接收的数据.

*高字节(HIGH BYTE)各位所代表意义

Bit 4 : 0 - NO OVERRUN , 1 - OVERRUN ERROR.

Bit 5 : 0 - NO PARITY ERROR , 1 - PARITY ERROR.

Bit 6 : 0 - NO FRAMING ERROR , 1 - FRAMING ERROR.

Bit 7 : 0 - NO RECEIVE BREAK , 1 - RECEIVE BREAK

■参阅函数

com_puts(),com_putb()

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        short i;
        char data;
        char msg[100];
        i=com_getc(2);
        if(i==(-1))sprintf(msg,"no data into com2 , rx_buffer is
empty.");
        else data=(i&0xff);
        com_puts(1,msg);
}
```

■摘要

```
#include <api.h>
short com_putb(port_no,block,size);
short port_no;   1->COM1 , 2->COM2
char *block;    区块的启始地址
short size;     区块的大小
```

■说明

把以block为启始地址,大小为size的一块区的资料,由传输port传输出去,在返回主程序后,须等到资料完全传输出去之后,才能再利用此函数.

注意:此函数为非等待模式(non-waiting)传输,当返回主程序时,传输仍在进行,当传输未完成时,资料区块(block)内容不能变更,否则传送的内容会与原预期有所出入.

■传回值

0 : 表示传输设定成功.

1 : 表示传输 port 正在传输资料中(正在执行上一次的传输命令).

■参阅函数

com_puts(),com_getc(),com_status()

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        char block[100];
        short size;
        short i;
        char msg[100];
        block="abcdefg";
        size=4;
        i=com_putb(2,block,size);
        if(i==0)sprintf(msg,"com2 transmit abcd ok.");
        else sprintf(msg,"com2 transmit abcd not ok.");
        com_puts(1,msg);
}
```

■摘要

```
#include <api.h>
short com_puts(port_no,string);
short port_no;    1->COM1 , 2->COM2
char *string     一串字符串
```

■说明

将所欲传输的字符串置入 TX_BUFFER 内,并由通讯 port 送出此字符串.

注意:此函数为非等待模式(non-waiting)传输,当返回主程序时,传输仍在进行,当传输未完成时,资料区块(block)内容不能变更,否则传送的内容会与原预期有所出入.

■传回值

0 : 表示已将所欲传输的字符串置入 TX_BUFFER 中,等待传输.

1 : 表示 TX_BUFFER 正在使用,无法把新的资料置入 TX_BUFFER 中,以作传输.

■批注

传输字符串须以 NULL(00H)为终结字符.

字符串最大长度 255 字.

■参阅函数

com_putb(),com_getc(),com_status

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        short i;
        char msg[100];
        i=com_puts(2,"string");
        if(i==0)sprintf(msg,"com2 transmit ok.");
        else sprintf(msg,"com2 transmit not ok.");
        com_puts(1,msg);
}
```

■摘要

```
#include <api.h>
short com_status(port_no);
short port_no;    1->COM1 , 2->COM2
```

■说明

核查传输 port 的 TX_BUFFER,RX_BUFFER,传输接收的状态.

■传回值

bit0　　0 : 表示 TX_BUFFER 尚有空间可容纳欲传输的资料.
　　　　1 : 表示 TX_BUFFER 已满.
bit2　　0 : 表示 RX_BUFFER 没有资料.
　　　　1 : 表示 RX_BUFFER 有资料.
bit3　　0 : 表示正在传输中.
　　　　1 : 表示传输完成.

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        short timer1;
        char msg[100];
        com_puts(2,"string");
        while(1)
        {
          if((com_status(2)&8)==8)
          {
              sprintf(msg,"com2 transmit ok.");
              break;
          }
        }
        timer1=set_timer(500)
        .
        .
}
```

■摘要

```
#include <api.h>
short go_rcv(port_no);
short port_no;   1->COM1 , 2->COM2
```

■说明

打开 RX_BUFFER 以备传输 port 所接收资料存放.

■传回值

无任何意义.

■参阅函数

stop_rcv()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        .
        .
        .
        i=com_init(2,&compbk);
        .
        .
        go_rcv(2);
        .
        .
}
```

■摘要

```
#include <api.h>
short stop_rcv(port_no);
short port_no;   1->COM1 , 2->COM2
```

■说明

关闭 RX_BUFFER,以免收到由传输 port 所接收非所欲的资料.

■传回值

无任何意义.

■参阅函数

go_rcv()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short i;
        char data;
        i=com_getc(2);
        .
        .
        stop_rcv(2);
}
```

■摘要

```
#include <api.h>
void cnfg_232_422(port_no);
short port_no;   1->COM1 , 2->COM2
```

■说明

设定传输 port 依据 DIP SW 的状态,来使用 RS232 或 RS422 来传输资料.

■传回值

无传回值

■批注

```
DIP SW1 0 : COM1 RS232
        1 : COM1 RS422
DIP SW2 0 : COM2 RS232
        1 : COM2 RS422
```

■参阅函数

cnfg_232(),cnfg_422()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        .
        .
        .
        i=com_init(2,&compbk);
        .
        .
        cnfg_232_422(2);
        .
        .
}
```

■摘要

```
#include <api.h>
void  cnfg_232(port_no);
short port_no;   1->COM1 , 2->COM2
```

■说明

设定传输 port 使用 RS232 来传输资料.

■传回值

无传回值

■参阅函数

cnfg_232_422(),cnfg_422()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        .
        .
        .
        i=com_init(2,&compbk);
        .
        .
        cnfg_232(2);
        .
        .
}
```

■摘要

```
#include <api.h>
void  cnfg_422(port_no);
short port_no;   1->COM1 , 2->COM2
```

■说明

设定传输 port 使用 RS422 来传输资料.

■传回值

无传回值

■参阅函数

cnfg_232_422(),cnfg_232()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        .
        .
        .
        i=com_init(2,&compbk);
        .
        .
        cnfg_422(2);
        .
        .
}
```

(2)与控制器沟通函数

| | |
|---|---|
| cdm_lock_read() | 读取 CDM 资料到 BUFFER,等待模式 |
| cdm_read() | 读取 CDM 资料到 BUFFER,不等待模式 |
| cdm_lock_wr() | 将 BUFFER 资料写到 CDM,等待模式 |
| cdm_wr() | 将 BUFFER 资料写到 CDM,不等待模式 |
| scan_data_in() | 将 SCAN DATA 输出至控制器 |
| scan_data_out() | 读取控制器输出的 SCAN DATA |

## CDM_LOCK_READ

■摘要

```
#include <api.h>
unsigned short cdm_lock_read(offset,size,buf)
unsigned short offset;      欲读取的CDM启始地址,0<offset<1920
unsigned short size;        欲读取的CDM的长度,单位为work, size < 64
word ,      size+offset< 1920 word
unsigned short  *buf;       读取CDM资料放置区,地址需为偶数
```

■说明

当CDM未被读取或写入资料时(等待模式),利用Lock read函数,可确保资料区块的完整性(即读取时区块内的资料不会一部份有被更新,一部份没被更新),可将CDM的启始地址为offset,大小为size的一区块的资料读回buf.

■传回值

1 : 读取成功

0 : 读取失败

■参阅函数

cdm_read(),cdm_lock_wr(),cdm_write()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short data[10]
        .
        cdm_lock_read(0,10,&data[0]);
        .
        .
}
```

■摘要

```
#include <api.h>
unsigned short cdm_read(offset,size,buf)
unsigned short offset;     欲读取的CDM启始地址,0<offset<1920
unsigned short size;       欲读取的CDM的长度 size < 64 word ,
size+offset< 1920 word
unsigned short  *buf;       读取CDM资料放置区,地址需为偶数
```

■说明

不管CDM是否被读取或写入资料时(不等待模式),将CDM的启始地址为offset,大小为size的一区块的资料读回buf.

■传回值

1 : 读取成功

0 : 读取失败

■参阅函数

cdm_lock_read(),cdm_lock_wr(),cdm_write()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short data[10]
        .
        cdm_read(0,10,&data[0]);
        .
        .
}
```

■摘要

unsigned short  *buf;        写入 CDM 资料放置区,地址需为偶数

■说明

当 CDM 未被读取或写入资料时(等待模式),利用此函数可避免在更新资料区块内容时,CPU 读取到不完整的资料, 将 buf 为启始地址,大小为 size 的一区块的资料写入 CDM 的 offset 为启始地址的区块.

■传回值

1 : 写入成功

0 : 写入失败

■参阅函数

cdm_lock_read(),cdm_read(),cdm_write()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short data[10]
        .
        data[0]=1;
        data[0]=2;
        data[0]=3;
        data[0]=4;
        data[0]=5;
        data[0]=6;
        cdm_lock_wr(0,6,&data[0]);
        .
        .
}
```

■摘要

```
#include <api.h>
unsigned short cdm_wr(offset,size,buf)
unsigned short offset;      欲写入的 CDM 启始地址,0<offset<1920
unsigned short size;        欲写入的 CDM 的长度 size < 64 word ,
size+offset< 1920 word
unsigned short  *buf;        写入 CDM 资料放置区,地址需为偶数
```

■说明

不管 CDM 是否被读取或写入资料时(不等待模式), 将 buf 为启始地址大小为 size 的一区块的资料写入 CDM 的 offset 为启始地址的区块.

■传回值

1 : 写入成功

0 : 写入失败

■参阅函数

cdm_lock_read(),cdm_read(),cdm_lock_wr()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short data[10]
        .
        data[0]=1;
        data[0]=2;
        data[0]=3;
        data[0]=4;
        data[0]=5;
        data[0]=6;
        cdm_write(0,6,&data[0]);
        .
        .
}
```

■摘要

```
#include <api.h>
void scan_data_in()
```

■说明

将 SCAN DATA 输出给控制器当输入.

■传回值

无传回值

■参阅函数

scan_data_out()

■范例

```
#include <api.h>
#include <stdio.h>

struct scan_data {
 UW di[2];
 UW ri[2];
 UW do[2];
 UW ro[2];
} test;

struct BD_DATA {
 UB      sub_id ;       /* Application board ID */
 UB      di_gp_cnt ;    /* DI 16 point word group cnt,0~8 */
 UB      ri_gp_cnt ;    /* RI 16 point word group cnt,0~8 */
 UB      do_gp_cnt ;    /* DO 16 point word group cnt,0~8 */
 UB      ro_gp_cnt ;    /* RO 16 point word group cnt,0~8 */
                        /* DI+RI+DO+RO <= 16 */
 void    *scan_data_ptr; /* pointer to the first variable of scan data
*/

} bd_dptb = { 1,2,2,2,2,&test};

main()
{
        .
        .
```

```
        test.di[0]=0x0f0f;
        test.di[1]=0xf0f0;
        test.ri[0]=0xff00;
        test.ri[1]=0x00ff;
        scan_data_in();
        .
        .
}
```

■摘要

```
#include <api.h>
void scan_data_out()
```

■说明

读取控制器的 Output SCAN DATA

■传回值

无传回值

■参阅函数

scan_data_in()

■范例

```
#include <api.h>
#include <stdio.h>

struct scan_data {
 UW di[2];
 UW ri[2];
 UW do[2];
 UW ro[2];
} test;

struct BD_DATA {
 UB      sub_id ;        /* Application board ID */
 UB      di_gp_cnt ;     /* DI 16 point word group cnt,0~8 */
 UB      ri_gp_cnt ;     /* RI 16 point word group cnt,0~8 */
 UB      do_gp_cnt ;     /* DO 16 point word group cnt,0~8 */
 UB      ro_gp_cnt ;     /* RO 16 point word group cnt,0~8 */
                         /* DI+RI+DO+RO <= 16 */
 void    *scan_data_ptr; /* pointer to the first variable of scan data
*/

} bd_dptb = { 1,2,2,2,2,&test};

main()
{
        UW a,b,c,d;
        .
```

```
        .
        scan_data_out();
        a=test.do[0];
        b=test.do[1];
        c=test.ro[0];
        d=test.ro[1];
        .
        .
}
```

(3)辅助函数

| | |
|---|---|
| timer() | 设定 10ms 的定时器 |
| timer_status() | 检查 10ms 的定时器 |
| set_timer() | 设定 1ms 的定时器 |
| chk_timer() | 检查 1ms 的定时器 |
| led_service() | 使 ACTIVE LED 闪烁 |
| brk() | 软件中断 |
| system() | 中止应用程序回到 Monitor |
| crc16() | 计算资料区的 CRC16 值 |

■摘要

```
#include <api.h>
short timer(timer_no,count);
short timer_no; 定时器号码(1-10)
short count;     计时时间以时基数计算,时基为 10ms
```

■说明

系统所提供 10ms 的定时器,作为计时用.

■传回值

0 : timer 设定完成

1 : timer 参数设定错误

■批注

每次调用定时器,定时器都重新计时,无论此定时器是否已开始计时.

■参阅函数

timer_status()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        short timer_no;
        short timer_cnt;
        timer_no=1; /* set timer 1 */
        timer_cnt=20; /* set time is 20*10=200ms */
        timer(timer_no,timer_cnt);
        .
        .
        .
}
```

■摘要

```
#include <api.h>
short  timer_status(timer_no);
short timer_no; 定时器号码,最大值为 10
```

■说明

核查所设定 10ms 的定时器,计时时间是否已结束.

■传回值

0 : 表示计时时间结束

1 : 表示计时时间尚未结束

■参阅函数

timer()

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        short timer_no;
        short timer_cnt;
        short i;
        char msg[100];
        timer_no=1; /* set timer 1 */
        timer_cnt=20; /* set time is 20*10=200ms */
        timer(timer_no,timer_cnt);
        i=timer_status(timer_no);
        if(i==0) sprintf(msg,"timer 1 : time is up.");
        else sprintf(msg,"timer 1 : time is not up.");
        com_puts(1,msg);
}
```

■摘要

```
#include <api.h>
unsigned long set_timer(time_ms);
unsigned short time_ms;        /* time base 1ms */
```

■说明

激活 1ms 定时器.

■传回值

timer handle

■参阅函数

chk_timer()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        int t1;
        t1=set_timer(1000); /* set 1 Sec timer start */
        .
        .
        .
}
```

■摘要

```
#include <api.h>
unsigned short chk_timer(hTIME);
unsigned long hTIME;
```

■说明

核查所设定 1ms 的定时器,计时时间是否已结束.

■传回值

0 : 表示计时时间结束

1 : 表示计时时间尚未结束

■参阅函数

set_timer()

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        int t1;
        t1=set_timer(1000); /* set 1 Sec timer start */
        while(1)
        {
          if(chk_timer(t1)=0) break;
        }
        .
        .
        .
}
```

■摘要

```
#include <api.h>
void led_service();
```

■说明

在一定时间内调用LED_SERVICE函数,可使面板上ACT灯号闪烁(5HZ),
否则将恒亮或恒灭,由此可知此片模块是否有运作.

注意:当使用多任务作业模式时请勿调用此函数.

■传回值

无传回值

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        while (1) {
        .
        .
        .
        led_service();
        .
        .
        }
 }
```

■摘要

```
#include <api.h>
void brk(brk_pnt);
char brk_pnt;     最多可设定 10 个断点(1-10)
```

■说明

软件中断功能.断点的操作请参考附录 1.

■传回值

无传回值

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
        .
        .
        .
        brk(1); /* 1 号断点 */
        .
        .
        brk(2); /* 2 号断点 */
        .
}
```

■摘要

```
#include <api.h>
void system();
```

■说明

停止应用程序,将控制权交回监督程序.

■传回值

无传回值

■范例

```
#include <api.h>
#include <stdio.h>
#include <string.h>

main()
{
        short i;
        char data;
        char msg[100];
        i=com_getc(2);
        if(i==(-1))sprintf(msg,"no data into com2 , rx_buffer is
empty.");
        else data=(i&0xff);
        com_puts(1,msg);
        system();
}
```

■摘要

```
#include <api.h>
unsigned short crc16(crc_init,data_buf,size);
unsigned short crc_init; /* crc16 的初始值 */
unsigned char  * buff;   /* 要计算 crc 的区块启始地址 */
unsigned short size;     /* 要计算 crc 的区块长度(byte) */
```

■说明

计算 CRC16 值.

■传回值

DATA BUFFER 的 CRC16 值.

■范例

```
#include <api.h>
#include <stdio.h>
main()
{
      char txbuff[100];
      unsigned short crc_check;
      txbuff[0] = 1;
      txbuff[1] = 3;
      txbuff[2] = 0;
      txbuff[3] = 0;
      txbuff[4] = 0;
      txbuff[5] = 10;
      crc_check=crc16(0xffff,txbuff,6);
      txbuff[6]=crc_check/0x100;
      txbuff[7]=crc_check%0x100;
      .
      .
      .
}
```

(4)多任务作业函数

| | |
|---|---|
| Create_task() | 建立一个新的 Task |
| yield_task() | 自动交出此 Task 的控制权给其它 Task |
| Task_get_pdata() | 取得 Create_task 时传进来的 data pointer |
| Task_set_priority() | 动态设定此 Task 的优先权 |
| Suspend_task() | 暂停某一特定 Task 的工作 |
| Resume_task() | 继续执行被暂停的 Task 工作 |
| preemtp_on() | 激活抢先式多任务 |
| preemtp_off() | 停止抢先式多任务 |
| Delay() | 将控制权交给其它 Task,等到延迟时间完成再继续执行 |
| Delay_until() | 将控制权交给其它 Task,(绝对时间)等到延迟时间完成再继续执行 |

■摘要

```
#include <api.h>
int Create_task(priority,stack_size,func_entry,private_data_ptr,
task_string);
unsigned short priority;          /* 设定此 Task 的优先权 0-15,0 最低
                                     15 最高 */
unsigned short stack_size;
/* 设定此 Task 所需 Stack 的大小 for function 的 automatic 变量*/15”高
void          *func_entry;        /* 设定此 Task 进入的地址 */
void          *private_data_ptr;  /* 设定此 Task 要传入的指针 */
unsigned char  *task_string;      /* 设定此 Task 名称以字符串表示 */
```

■说明

建立一个新的 Task.

■传回值

task handle

■参阅函数

Task_get_pdata(),Task_set_priority()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task1(),&port2,"APP TASK #2");
}
app_task1()
{
   .
   .
}
```

## YIELD_TASK

■摘要

```
#include <api.h>
void yield_task();
```

■说明

自动交出此 Task 的控制权给其它 Task.

■传回值

无传回值

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task1(),&port2,"APP TASK #2");
}
app_task1()
{
 while(1)
 {
    .
    .
    yield_task();
    .
 }
}
```

■摘要

```
#include <api.h>
 Task_get_pdata();
```

■说明

取得 Create_task 时传进来的 data pointer.

■传回值

data pointer

■参阅函数

Create_task()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task1(),&port2,"APP TASK #2");
}
app_task1()
{
 int port_no;
 port_no = *(int *)Task_get_pdata();
 .
 .
 .
}
```

■摘要

```
#include <api.h>
void Task_set_priority(new_priority);
unsigned short new_priority;        /*优先权 0-15 */
```

■说明

动态设定此 Task 的优先权.

■传回值

无传回值

■参阅函数

Create_task()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task1(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 .
 Task_set_priority(6);
 .
}
```

## SUSPEND_TASK

■摘要

```
#include <api.h>
int  Suspend_task(task_handle);
int  task_handle;
```

■说明

暂停某一特定 Task 的工作.

■传回值

0:ok

1:error

■参阅函数

Resume_task()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
task_main()
{
 int task1;
 comm_init();
 task1=Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 .
 .
 Suspend_task(task1);
}
app_task1()
{
 .
 .
 .
 .
}
```

■摘要

```
#include <api.h>
int  Resume_task(task_handle);
int  task_handle;
```

■说明

继续执行被暂停的 Task 工作.

■传回值

0:ok

1:error

■参阅函数

Suspend_task()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
task_main()
{
 int task1;
 comm_init();
 task1=Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 .
 .
 Suspend_task(task1);
 .
 .
 Resume_task(task1);
}
app_task1()
{
 .
 .
 .
 .
}
```

## PREEMTP_ON

■摘要

```
#include <api.h>
void preemtp_on();
```

■说明

以抢先式多任务,最高 priority 的每一 Task 轮流 10ms 执行一次(如未主动交出).

■传回值

无传回值

■参阅函数

preemtp_off()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task2(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 while(1)
 {
   .
   preemtp_on();
   .
 }
 .
}
app_task2()
{
 .
}
```

## PREEMTP_OFF

■摘要

```
#include <api.h>
void  preemtp_off();
```

■说明

停止抢先式多任务.

■传回值

无传回值

■参阅函数

preemtp_on()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task2(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 .
 .
 Preemtp_off();
 .
 .
}
app_task2()
{
 .
 .
 .
 .}
```

■摘要

```
#include <api.h>
void Delay(time_10ms);
unsigned short time_10ms;       /* time base 10ms */
```

■说明

作延迟的动作,时间未到时会将控制权交给其它 Task.

■传回值

无传回值

■参阅函数

Delay_until()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task2(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 .
 while(1)
  {
    com_puts(1,"This is a test for com1");
    Delay(100);
  }
}
app_task2()
{
 .
 .
}
```

■摘要

```
#include <api.h>
void Delay_until(abs_time);
unsigned short abs_time;        /* time base 10ms */
```

■说明

作延迟的动作,时间未到时会将控制权交给其它 Task.利用此函数可做到不会只累积误差的固定时距动作(此时间为绝对时间).

■传回值

无传回值

■参阅函数

Delay(),Get_Timer()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
UL abs_time;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task2(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 abs_time=Get_timer();
 while(1)  /*每一秒送出”This is...”字符串一次*/
 {
    com_puts(1,"This is a test for com1");
    abs_time+=100;
    Delay_until(abs_time);
 }
}
app_task2()
{
.
.
}
```

■摘要

```
#include <api.h>
UL Get_timer(void)
```

■说明

取得系统的绝对时间

■传回值

系统的绝对时间

■参阅函数

Delay_until()

■范例

```
#include <api.h>
#include <stdio.h>
int port1=1;
int port2=2;
UL abs_time;
task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),&port1,"APP TASK #1");
 Create_task(8,0x100,&app_task2(),&port2,"APP TASK #2");
}
app_task1()
{
 .
 abs_time=Get_timer();
 while(1)  /*每一秒送出" This is..." 字符串一次*/
  {
    com_puts(1,"This is a test for com1");
    abs_time+=100;
    Delay_until(abs_time);
 }
}
app_task2()
{
.
}
```

# 附

# 录

# 一

德维森公司提供一个 Program Loader 及 Debug Tool 程序 Hpcmoni.exe 供 HLM20 使用者开发程序使用,现将其用法叙述如下:
使用方法:由 PC 的 COM1 或 COM2 经 RS232 连接至 HLM20 的 COM1.
语法: HPCMONI 1 由 COM1 与 HLM20 连接
HPCMONI 2 由 COM2 与 HLM20 连接
由 COM1 连接 HLM20 Module 可以用以下的命令,进入后显示”>”表示联机成功.

```
D:MCC68K>HPCMONI 1
PC Monitor for HLM20 Module Ver. 2.0 (c) 10/21/1998 TECHWAYSON INDUSTRIAL LTD.
Press ? to see help
Select COM #1 Baud = 38400

>
```

图 1　进入 HPCMONI 的画面

Hpcmoni.exe 提供以下功能:

1. Down Load User Program
2. Load Symbol Table
3. Display Memory Content
4. Modify Memory Content
5. Fill Memory Content
6. Run Program
7. Reset Program
8. Write User Program into ROM
9. Set/Display Break Point
10. Word Mode Access
11. Byte Mode Access
12. Quit From Monitor
13. File Directory Display
14. Monitor version display
15. Process status
16. Help
17. Other instruction

1. Help

由 HLM20 模块将指令说明送到 Monitor 供 User 参考.

**语法:　　?**

```
>?
=======   HLM20 monitor command summary   ======

 Down Load User Program           DO   [File_Name]
 Load Symbol Table                LO   File_Name
 Display Memory Content           D    ADDR [SIZE]
 Modify  Memory Content           M    ADDR DATA
 Fill    Memory Content           F    ADDR DATA SIZE
 Run     Program                  G
 Reset   Program                  X
 Write User Program into ROM      P
 Set/Display Break Point          H    [Brk_No Loop_cnt ]
 Word Mode Access                 W
 Byte Mode Access                 B
 Quit From Monitor                Q
 File Directory Display           DIR File_Spec
 Monitor version display          V
 Process status                   SS
 Set Module ID                    ID
 Rententive memory test           MT
 Examine front switch             SW
 Write system monitor to ROM      PS
 Help                             ?
>
```

图 2　Help 的画面

2. Down Load User Program

将使用者开发好的应用程序(*.ABS)及 Symbol Table(*.MAP) 下载到 HLM20 模块.

**语法:　　DO　[File_Name]**

**File_Name 可含路径,不含扩展名**

```
>DO EXAMPLE
                下载正在进行!

022661
>                下载成功!
            Total 72 symbols loaded!
(Down Load User Program 成功)
>
>DO ABC
file <ABC> not found !
Symbol file <ABC.map> not found !
(Down Load User Program 失败)
>
```

图 3　Down Load User Program 的画面

3. Load Symbol Table

将使用者开发好应用程序的 Symbol Table(*.MAP)下载至 HLM20 模块,以搭配 Display Memory Content 指令使用.

**语法:　　LO　File_Name**

**File_Name 可含路径,不含扩展名**

```
>LO EXAMPLE
            Total 72 symbols loaded!
(Load Symbol Table 成功)
>
>LO ABC
Symbol file <ABC.map> not found !
(Load Symbol Table 失败)
>
```

图 4　Load Symbol Table 成功的画面

4. Word Mode Access

将 Memory 显示模式设定为 Word 模式.

**语法:**　　W

```
>W
Set to word mode
>D 80100 10
          0000 0002 0004 0006 0008 000A 000C 000E  ASSCII-CODE
080100    0000 0000 0000 0000 0000 0000 0000 0000 ···············.
>
```

图 5　Word Mode 寄存器显示画面

5. Byte Mode Access

将 Memory 显示模式设定为 Byte 模式.

**语法:**　　B

```
>B
Set to byte mode
>D 80100 10
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ASSCII-CODE
080100    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ···············.
>
```

图 6　Byte Mode 寄存器显示画面

6.Display Memory Content

将内存的内容输出到屏幕上.

**语法:** D ADDR [SIZE]

说明:ADDR 可为绝对地址或当有 Load Symbol Table 时可为 Global Variable 的名称.SIZE 可有可无(单位为 BYTE,16 进制).

```
>B
Set to byte mode
>D 80100 10
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ASSCII-CODE
080100    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············.
>D .i
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ASSCII-CODE
060330               01 02 03 04 00 00 00 00 00 00 00 00      .·········..
>
```

图 7 寄存器显示画面

各种资料型别所需寄存器大小如下:

```
typedef  unsigned char     UB;      /*  8 Bit */
typedef  unsigned char     BYTE;    /*  8 Bit */
typedef  unsigned short    UW;      /* 16 Bit */
typedef  unsigned short    WORD;    /* 16 Bit */
typedef  unsigned long     UL;      /* 32 Bit */
typedef  unsigned int      LONG;    /* 32 Bit */
typedef  short             INT16;   /* 16 Bit */
typedef  unsigned short    UINT16;  /* 16 Bit */
typedef  int               INT32;   /* 32 Bit */
typedef  unsigned int      UINT32;  /* 32 Bit */
```

若下达 d .i 指令得到上图的结果

char  i 则 i=0x01

short i 则 i=0x102

int   i 则 i=0x1020304

long  i 则 i=0x1020304

7. Modify  Memory Content

更改内存内容.

**语法:**  M  ADDR DATA

说明:若设定为 Byte Mode 则更改一个 Byte 的资料，若设定为 Word Mode 则更改一个 Word 的资料.

```
>B
Set to byte mode
>D 80100 10
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ASSCII-CODE
080100    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
>M 80100 1
>D 80100 10
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ASSCII-CODE
080100    01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
>W
Set to word mode
>M 80100 1
>D 80100 10
          0000 0002 0004 0006 0008 000A 000C 000E  ASSCII-CODE
080100    0001 0000 0000 0000 0000 0000 0000 0000  ...............
>
```

图 8　更改寄存器内容画面

8. Fill Memory Content

将一个内存区块全部填入相同的数值.

**语法:F ADDR DATA SIZE (SIZE<=100H,Byte 为单位)**

```
>F 80100 FF 10

>D 80100 10
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ASSCII-CODE
080100    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ················.
>W
Set to word mode
>F 80100 FF 10

>D 80100 10
          0000 0002 0004 0006 0008 000A 000C 000E  ASSCII-CODE
080100    00FF 00FF 00FF 00FF 00FF 00FF 00FF 00FF  ················.
>
```

图 9 更改寄存器区块内容画面

9. Run Program

执行下载到 HLM20 的应用程序.

**语法: G**

```
>DO EXAMPLE
                Down Load in Progress

022661
>                Down Load Complete!
            Total 72 symbols loaded!

>G
User Program Start …
                       此时扳动面板上的 Toggle SW 中断程序执行
Program Broke by key
PC=000225EE
>
```

图 10 执行应用程序的画面

10. Reset　Program

将 Program Counter 指向应用程序起点.

**语法:**　　X

```
>X
User Program Restart !
>
```

图 11　Reset Program 的画面

11. Write User Program into ROM

将下载到 HLM20 的应用程序烧到 FLASH ROM 的内.

(将面板上的 Toggle SW 置于上方,才可作烧录的动作)

**语法:**　　P

```
>P                    Toggle SW is down
Programing Start .
Program Fail!
>P                    Toggle SW is up
Programing Start ……………………………………..
Program OK !
>
```

图 12　Write User Program 的画面

12. File Directory Display

与 DOS 的 DIR 指令相同

**语法:**　　DIR File_Spec

13.Monitor version display

显示 HLM20 模块内部,Monitor 程序版本.

**语法:**　　V

```
>V
HLM20 Module Monitor Ver. 4.2 (c) 12/15/1998 TECHWAYSON INDUSTRIAL LTD.
>
```

图 13　Monitor version display 的画面

14. Set/Display Break Point

设定或显示断点的状态.

**语法:** H [Brk_No Loop_cnt ]

Brk_No:断点号码 1-10

Loop_cnt:0-65535

```
>H              显示断点的设定状态
BRK(1)=0        0:断点 Disable
BRK(2)=0
BRK(3)=0
BRK(4)=0
BRK(5)=0
BRK(6)=0
BRK(7)=0
BRK(8)=0
BRK(9)=0
BRK(10)=0

>H 1 10

>H              显示断点的设定状态
BRK(1)=10       1 号断点 Enable,程序执行经过 10 次会将控制权交给 Monitor
BRK(2)=0
BRK(3)=0
BRK(4)=0
BRK(5)=0
BRK(6)=0
BRK(7)=0
BRK(8)=0
BRK(9)=0
BRK(10)=0

>G
User Program Start …
 Program Break at BRK(1)
>
```

图 14 Set/Display Break Point 的画面

15. Quit From Monitor

退出 Monitor 程序返回 DOS.

**语法: 1.出现”>”符号,Q**

**2.未出现”>”符号,”ESC”**

```
>Q

D:MCC68K>

或
>G
User Program Start …

此时按下ESC键即可返回DOS
D:\MCC68K>
```

图 15　Quit From Monitor 的画面

16.Process status

检查 Muti-tasking 状态用

**语法:　　SS**

```
Program Broke by key
PC = 00022BF0
>SS
                    Priority    Stack_status
Task ID   State      R    B      MAX   USE      Task Name
-------   -------  ----------   ------------  -----------------------
 0040       RUN     00   00      0100  006a   Null Task
 0080        WT     08   08      0200  0082   RUN LED TASK
 00c0        WT     08   08      0200  0106   APP TASK #1
 0100        WT     08   08      0200  0134   APP TASK #2
>
```

图 16　Process status 的画面

17.其它指令

Set Module ID

Rententive memory test

Examine front switch

Write system monitor to ROM

以上 4 个指令为生产测试用,用户切勿使用

# 附

# 录

# 二

功能说明:由 HLM20 端口 2 以 9600,E,8,2,MOD BUS 的格式读取控制器保持缓存器的值,启始地址由 SCAN DATA RO[1] 设定(40001-49999),长度由 SCAN DATA RO[2]设定(Length<125),读回来的值放在 CDM 地址为 SCAN DATA RO[3](CDM Addr+Length>1920). ERROR CODE 放在 SCAN DATA RI[1](0 READ OK,1 TIME OUT,2 CRC ERROR,3 SETTING ERROR).

| ID | FUNCTION CODE | Start Addr HI | Start Addr LO | DATA OF REG S HI | DATA OF REG S LO | CRC 16 HI | CRC 16 LO |
|---|---|---|---|---|---|---|---|
| 01 | 03 | 00 | 00 | 00 | 02 | ?? | ?? |

读保持缓存器的询问信息

| ID | FUNCTION CODE | BYTE COUNT | DATA HI 40001 | DATA LO 40001 | DATA HI 40002 | DATA LO 40002 | CRC 16 HI | CRC 16 LO |
|---|---|---|---|---|---|---|---|---|
| 01 | 03 | 04 | 00 | 01 | 00 | 02 | ?? | ?? |

读保持缓存器的响应信息

```
#include <stdio.h>
#include <confg.h>
#include <api.h>

struct INIT {
UW        err;
UW        start_addr;
UW        length;
UW        cdm_addr;
} test;
struct BD_DATA {
 UB       sub_id ;             /* Application board ID */
 UB       di_gp_cnt ;          /* DI 16 point word group cnt,0~8 */
 UB       ri_gp_cnt ;          /* RI 16 point word group cnt,0~8 */
 UB       do_gp_cnt ;          /* DO 16 point word group cnt,0~8 */
 UB       ro_gp_cnt ;          /* RO 16 point word group cnt,0~8 */
                               /* DI+RI+DO+RO <= 16 */
 void     *scan_data_ptr;      /* pointer to the first variable of scan
                                  data */
} bd_dptb = { 7,0,1,0,3,&test};
```

```
struct COMM {
UB   id;
UB   port;
UW   start_addr;
UW   length;
UW   rd_cdm_addr;
UW   wr_cdm_addr;
UB   txbuff[300];
UB   rxbuff[300];
UW   read_data[125];
UW   write_data[125];
} ;

union DATA_CONVERT {
         UW integer;
         UB ch[2];
} ;
struct COM_PMBK  {
    char           baud;
    char         parity;
    char       data_bit;
    char       stop_bit;
    char        tx_ctrl;
    char        rx_ctrl;
    char       com_mode;
    char       tmo_mode;
    char        tmo_cnt;
    };

struct COM_PMBK compbk ;

void comm_init1()
{
 compbk.baud = BAUD_38400;
 compbk.parity = PARITY_EVEN;
 compbk.data_bit = DBIT_8;
 compbk.stop_bit = STB_2;
```

```
 compbk.tx_ctrl = 0;
 compbk.rx_ctrl = 0;
 compbk.com_mode= 0;
 compbk.tmo_mode= 0;
 com_init(1,&compbk);
 cnfg_232(1); /* set 232 */

}
void comm_init2()
{
 compbk.baud = BAUD_9600;
 compbk.parity = PARITY_EVEN;
 compbk.data_bit = DBIT_8;
 compbk.stop_bit = STB_2;

 compbk.tx_ctrl = 0;
 compbk.rx_ctrl = 0;
 compbk.com_mode= 0;
 compbk.tmo_mode= 0;
 com_init(2,&compbk);
 cnfg_232(2); /* set 232 */
}

main()
{
        UW start_addr,length,cdm_addr;
        struct COMM PPC;
        while(1)
        {
           led_service();
           scan_data_out();
           start_addr=test.start_addr;
           length=test.length;
           cdm_addr=test.cdm_addr;

if(start_addr>40000&&(start_addr+length)<50000&&length<125
           &&length>0&&(cdm_addr+length)<1920)
```

```
                test.err=read_4xxxx(start_addr,length,cdm_addr);
            else
                test.err=3;
            scan_data_in();
        }
}

read_4xxxx(r_addr,length,cdm_addr)
UW r_addr;
UW length;
UW cdm_addr;
{
        UW err,i;
        struct COMM PPC;
        PPC.id=1;
        PPC.port=2;
        PPC.start_addr=r_addr;
        PPC.length=length;
        PPC.wr_cdm_addr=cdm_addr;
        err=read_PPC4xx(&PPC);
        return(err);
}

read_PPC4xx(buffer)
struct COMM *buffer;
{
        UB retry;
        UW i,error,status;
        UW rxc,crc_check;
        UL timer1,timer2;
        UB msg[100];
        union DATA_CONVERT convert;
        buffer->txbuff[0]=buffer->id;
        switch(buffer->start_addr/10000)
        {
           case  3 :  buffer->txbuff[1] = 4; break;
           case  4 :  buffer->txbuff[1] = 3; break;
        };
```

```
buffer->txbuff[2]=((buffer->start_addr%10000)-1)/0x100;
buffer->txbuff[3]=((buffer->start_addr%10000)-1)%0x100;
buffer->txbuff[4]=0;
buffer->txbuff[5]=buffer->length;
crc_check=crc16(0xffff,buffer->txbuff,6);
buffer->txbuff[6]=crc_check/0x100;
buffer->txbuff[7]=crc_check%0x100;
retry=0;
while(retry<2)
{
        retry++;
        comm_init2();
        com_putb(buffer->port,buffer->txbuff,8);
        go_rcv(buffer->port);
        i=0;
        while(1)
        {   /* check transmit ok. */
            if((com_status(buffer->port)&0x8)==0x8) break;
        }
        timer2=set_timer(500);
        while(1)
        {
                led_service();
                rxc=com_getc(buffer->port);
                if(rxc != -1)
                {
                        /* charater interval */
                        timer2=set_timer(100);
                        buffer->rxbuff[i]=rxc&0xff;
                        i++;
                        if(i>(4+buffer->length*2))
                        {
                                /* check receive data crc16 */
                                if(crc16(0xffff
                                       ,buffer->rxbuff,i)==
0)
                                        error=0;
```

```
                                        else
                                               error=2; /* crc error
*/
                                        break;
                                }
                        }
                        if(chk_timer(timer2)==0)  /* time out */
                        {
                                error=1;
                                break;
                        };
                } /* while(1) */
                if(error==0) break;
        } /* while(retry<2) */
        comm_init1();
        if(error==0)
        {
            for(i=0;i<buffer->length;i++)
            {
                    convert.ch[0]=buffer->rxbuff[3+i*2];
                    convert.ch[1]=buffer->rxbuff[4+i*2];
                    buffer->read_data[i]=convert.integer;
                    buffer->write_data[i]=convert.integer;
            }
            if(buffer->length>64)
            {
                cdm_lock_wr(buffer->wr_cdm_addr,64
                ,buffer->read_data);

cdm_lock_wr(buffer->wr_cdm_addr+64,buffer->length-64,
                &buffer->read_data[64]);
            }
            else
                cdm_lock_wr(buffer->wr_cdm_addr,buffer->length
                ,buffer->read_data);
            sprintf(msg,"Read %05d - %05d to CDM %04d ok.\r\n"
            ,buffer->start_addr,buffer->start_addr+buffer->length-1
            ,buffer->wr_cdm_addr);
```

```
            com_puts(1,msg);
            timer1=set_timer(300);
            while(chk_timer(timer1)!=0)led_service();
        }
        else
        {
            sprintf(msg,"Read %05d - %05d to CDM %04d time out.\r\n"
            ,buffer->start_addr,buffer->start_addr+buffer->length-1
            ,buffer->wr_cdm_addr);
            com_puts(1,msg);
            timer1=set_timer(3000);
            while(chk_timer(timer1)!=0)led_service();
        }
        return(error);
}
```

# 附

# 录

# 三

```
#include          <stdio.h>
#include          <string.h>
#include          <confg.h>
#include          <api.h>

struct BD_DATA {
 UB      sub_id ;               /* Application board ID */
 UB      di_gp_cnt ;            /* DI 16 poshort word group cnt,0~8 */
 UB      ri_gp_cnt ;            /* RI 16 poshort word group cnt,0~8 */
 UB      do_gp_cnt ;            /* DO 16 poshort word group cnt,0~8 */
 UB      ro_gp_cnt ;            /* RO 16 poshort word group cnt,0~8 */
                                /* DI+RI+DO+RO <= 16 */
 void    *scan_data_ptr;        /* poshorter to the first variable of
scan data */

} bd_dptb = { 0,0,0,0,0,0};

struct COM_PMBK  {
    char           baud;
    char          parity;
    char        data_bit;
    char        stop_bit;
    char         tx_ctrl;
    char         rx_ctrl;
    char        com_mode;             /* <=2 */
    char        tmo_mode;             /* <=2 */
    char         tmo_cnt;
    };

struct COM_PMBK compbk ;

void comm_init()
{
 compbk.baud = BAUD_38400;
 compbk.parity = PARITY_EVEN;
 compbk.data_bit = DBIT_8;
 compbk.stop_bit = STB_1;
```

```
 compbk.tx_ctrl = 0;
 compbk.rx_ctrl = 0;
 compbk.com_mode= 0;
 compbk.tmo_mode= 0;
 com_init(1,&compbk);
 com_init(2,&compbk);

}

task_main()
{
 comm_init();
 Create_task(8,0x100,&app_task1(),1,"APP TASK #1");
 Create_task(8,0x100,&app_task1(),2,"APP TASK #2");
}

app_task1()
{
 int port_no;
 int delay_time;
 int i=0;
 UB msg[100];
 port_no = Task_get_pdata();
 if(port_no==1) delay_time=100;
 else delay_time=50;
 while(1)
 {
    sprintf(msg,"This is a test for com%d , count=%d\n\r",port_no,i);
    com_puts(port_no,msg);
    Delay(delay_time);
/*  yield_task();*/
    i++;
 }
}
```




Http://www.techwayson.com

德维森科技(深圳)有限公司

中国深圳市南山区高新区科技南12路照明中心北二楼

Tel: 0755-26715433 Fax: 0755-26715422 P.C.: 518057